Fast Private Association Rule Mining by A Protocol for Securely Sharing Distributed Data

Vladimir Estivill-Castro  
School of ICT, Griffith University, Queensland, Australia 4111  
Email: v.estivill-castro@griffith.edu.au

Ahmed HajYasien  
Faculty of Engineering Amman University, Amman, Jordan  
Email: ahajyasien@ammanu.edu.jo

Abstract—Privacy concerns may discourage users who would otherwise join beneficial data mining tasks for intelligence and/or security. We propose an efficient protocol that allows parties to share data in a private way with no restrictions and without loss of accuracy. Our method has the immediate application that horizontally partitioned databases can be brought together and made public without disclosing the source/owner of each record. At another level, we have an additional benefit that we can apply our protocol to privately discover association rules. Our protocol is more efficient than previous methods. The effects of our protocol are less than others: 1) each party can identify only their data, 2) no party is able to learn the links between other parties and their data, 3) no party learns any transactions of the other parties' databases.

I. INTRODUCTION

Intelligence usually refers to the gathering and analysis of information to gain knowledge. In the current global climate, Knowledge Discovery and Data Mining have become central tools of this process [1]. Moreover, it is commonly the case that data-collection is performed by different agencies, organisations or governments with different jurisdictions and mandates. Keeping the data of each party private while they engage in collaborative intelligence or data analysis exercises is an essential aspect of security as well. Recent meetings on Intelligence and Security Informatics directly addressed collaborative data mining that ensures privacy [2]-[4]. Among the core data mining tasks is association-rule mining.

In has been argued that collaboration for association-rule mining between different agencies could hold the key to effective homeland security as these parties combine strengths in different areas [5]. Association-rule mining in the context of several parties each holding some data is attractive in several privacy settings [6], [7]. When parties hold different attributes on common transactions is referred as mining a vertically partitioned database. Under this argument, several studies have proposed methods and approaches for association-rule mining for vertically partitioned data [5], [8]. However, the other common case is horizontally partitioned data, where different jurisdictions, or enterprises on a common domain, hold common attributes but about different transactions. For example, mobile phone companies hold very similar attributes about their customers and their accounts, although they may have a very different and almost disjoint set of customers (and certainly transactions). However, attributes like length of the call, age of the customer, and so on will be common. Similar for government agencies, financial institutions, and any group of organisations that operate on a common domain. This case has been studied much less, with essentially one method introduced in 2002 [9] and slightly refined later [10].

The idea that data owners on a common domain share their data for the common good collides with privacy rights and the need for security. Privacy concerns have been influencing data owners and preventing them from achieving maximum benefits with data sharing. One common alternative is for data owners to sanitise their data and try to block as many inference channels as possible to prevent other parties from concluding what they consider sensitive. Data sanitization is defined as the process of making sensitive information in non-production databases safe for wider visibility [11] and is practised as a main alternative to balancing privacy and security in collaborative exercises for association-rule mining.

In this paper, we use the alternative to data sanitization, usually referred as privacy-preserving data mining. In this approach, the parties collaborate to a data mining task that will use as input the union of all the data sets of the interested parties. They all will receive and benefit from the result, and that is why they contribute their data. However, the challenge is that no party learns much about another’s party data except what can be inferred from the output. Many researchers have produced algorithms for several data-mining tasks, like classification (with decision trees [12] or by k-nearest neighbours [13]), or for clustering [14], [15]. However, many of these algorithms have not found implementations because most of them rely at some point on a theorem that says that a function f can be implemented in secure multi-party computation (see Sec II-B) if it can be lay out as a circuit of polynomial size. Coming up for such circuits, even for simple functions, has proven challenging. The next difficulty is that the protocol is derived from splitting shares of the inputs and outputs of the circuit’s gates.

We address a core and important data mining task and produce an efficient and practical implementation. We solve the issue of association-rule mining for horizontally-partitioned data with a more efficient and implementable protocol. In the next section we provide details of the context for this work. We highlight some of the disadvantages of the data sanitization
approach and its variants. Preliminaries are presented to understand the context of our work before some new protocols. We then discuss the improvement the new protocol represents over previous work.

II. PRELIMINARIES

A. Data Sanitization

Oliveira et al. [16] classified the existing sanitising algorithms into two major classes: data-sharing techniques and pattern-sharing techniques. Data-sharing techniques communicate data to other parties without analysis or summarisation by data mining or statistical techniques. Under this approach, a level of security is achieved by algorithms that change databases and produce distorted databases in order to hide sensitive data. Data-sharing techniques are, in themselves, categorised as follows. First, (item restriction)-based: algorithms. In this class, the methods [11], [17]–[20] reduce either the support or confidence to a safe zone (below a given privacy support threshold) by deleting transactions or items from a database to hide sensitive rules that can be derived from that database. Second, (item addition)-based: algorithms. This group of methods [17] add imaginary items to the existing transactions. Usually the addition is performed to items in the antecedent part of the rule. Thus, the confidence of such a rule is reduced and enters the safe zone. The problem with this approach is that the addition of new items will create new rules and parties could share untrue knowledge (sets of items that are not frequent itemsets appear as such). Third, (item obfuscation)-based: algorithms. The algorithms [21] replace some items with a question mark in some transactions to avoid the exposure of sensitive rules. Unlike the (item addition)-based, the (item obfuscation)-based, approach saves parties from sharing false rules.

II. This second major class is pattern-sharing techniques, where the sanitising algorithms act on the rules mined from a database, instead of the data itself. The existing solutions either remove all sensitive rules before the sharing process [16] (such solutions have the advantage that two or more parties can apply them) or share all the rules in a pool where no party is able to identify or learn anything about the links between individual data owned by other parties and their owners [9] (such solutions have the disadvantage that they can be applied only to three or more parties).

Data sanitization essentially correspond to the classical techniques of Data Perturbation [22]–[24], and as it has been pointed before, they always trade security for less confidence and accuracy in the results. Thus, when data owners share inaccurate data, they will end up with inaccurate results. And since data owners are willing to share their data for the aim of enhancing their knowledge and intelligence, results with poor confidence will guide them to incorrect decisions that might have negative effects. The balance between privacy and accuracy on data-perturbation techniques depends on modifying the data in a way that no party can reconstruct data of any individual transaction but the overall mining results are somewhat valid and close to the exact ones. In other words, the more the distortion to block more inference channels, the less accurate the result will be. In general, it has been demonstrated that in many cases random data distortion preserves very little privacy [25].

The alternative to providing distorted/noisy data is to provide the actual data. We achieve our claim for association-rule mining on the union of the parties’ data by introducing a protocol that hides the link between each data record and its owner. There seem to be a contradiction in the previous statement. This paper introduces a protocol where parties can share their data without distortion and no party can link data to a specific party. Our protocol will improve upon the protocol presented in [9]. We remove the overhead to the mining task and reduce the number and complexity of steps needed. The advantages of our protocol are: 1) there is no loss of accuracy 2) there are no limitations on what each party can do privately with the shared data. A disadvantage of this protocol is that it can be conducted only on three or more parties.

B. Secure Multi-Party Computation

Researchers have proposed algorithms for privacy-preserving data mining under the model named secure multi-party computation [26]–[31]. This approach uses cryptographic tools to the problem of computing a data-mining task from distributed data sets, while keeping local data private [32]–[37]. These tools allow parties to analyse their data and achieve results without disclosure of the actual data. For privacy-preserving association rule-mining in horizontal data, Kantarcioglu et al. [9] proposed a method that incorporates cryptographic techniques and show frequent itemsets, of three or more parties, as one set where each party can recognise its itemsets but can not link any of the other itemsets to their owners. Although refined later [10], this particular method uses commutative encryption, which could be very expensive if we have a large number of parties.

More formally, in secure multi-party computation p parties, \(P_0, \ldots, P_p\) wish to evaluate a function \(F(\bar{x}_1, \ldots, \bar{x}_p)\), where \(\bar{x}_i\) is a secret vector of values provided by \(P_i\). The goal is to preserve the privacy of the each party’s inputs and guarantee the correctness of the computation. This problem is trivial if we add a trusted third party \(T\) to the computation. Simply, \(T\) collects all the inputs from the parties, computes the function \(F\), and announces the result. If the function \(F\) to be evaluated is a data mining task, we call this privacy-preserving data mining in secure multi-party computation (PPDM-SMC).

It is usually difficult to agree on a trusted party. The algorithms proposed to solve privacy-preserving tasks usually assume no trusted party, but assume a semi-honest model. The semi-honest model is a more realistic abstraction of how parties would engage and participate in a collective computation while preserving each the privacy of their data.

C. The semi-honest model

A semi-honest party is one who follows the protocol steps but feels free to deviate in between the steps to gain more knowledge and satisfy an independent agenda of interests. In
other words, a semi-honest party follows the protocol step by step and computes what needs to be computed based on the input provided from the other parties, but it can do its own analysis during or after the protocol to compromise privacy/security of other parties. It will not insert false information that will result in failure to compute the data mining result, but will use all the information gained to attempt to infer or discover private values from the data sets of other parties. Semi-honest parties will not collude with others or release partial results that the protocol determines as private.

A definition of security under the semi-honest model [38] formalises that whatever a semi-honest party learns from participating in the protocol, this information could be essentially obtained from its inputs and its outputs. In particular, the definition models the computation with a probabilistic function $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$ computable in polynomial-time. Here, $f_1(x,y)$ denotes the first element of $f(x,y)$, and says that what the output string is for the first party as a function of the inputs strings $x$ and $y$ of the two parties (and $f_2(x,y)$ is the respective second component of $f(x,y)$ for the second party). The two-party protocol is denoted by $\Pi$. The view of the first party during an execution of $\Pi$ on $(x,y)$ is denoted by $\text{VIEW}^f_1(x,y)$. Such view consists of $(x,r,m_1,\ldots,m_k)$, where $r$ represents the outcome of the first party’s internal coin tosses and $m_i$ represents the $i^{th}$ message it has received. Then, we define that $\Pi$ can privately compute $f$, with respect to the first party, if there exist a probabilistic polynomial time algorithms $S_1$ such that even if party two provides arbitrary answers during the protocol, the corresponding view for the first party is the output of the algorithm $S_1$ on the input $x$ of the first party and the messages received by the first party. The protocol can privately compute $f$ if it can do so with respect to both parties. The theory of this model [38] shows that to compute privately under the semi-honest model is also equivalent to compute privately and securely.

A fundamental tool to construct private and secure protocols as defined above is a public-key crypto-system.

D. Public-key Crypto-systems (asymmetric ciphers)

Public-key crypto-system were invented in the late 1970’s, along developments in complexity theory [39], [40]. As a result, crypto-systems could be developed which would have two keys, a private key and a public key. With the public key, one could encrypt data, and decrypt them with the private key. Thus, the owner of the private key would be the only one who could decrypt the data, but anyone knowing the public key could send them a message in private. Many of the public key systems are also patented by private companies, this also limits their use. For example, the RSA algorithm was patented by MIT in 1983 in the United States of America as (U.S. patent #4,405,829). The patent expired on 21 September 2000.

The RSA algorithm was described in 1977 [41] by Ron Rivest, Adi Shamir and Len Adleman at MIT; the letters RSA are the initials of their surnames. RSA is currently the most important public-key algorithm and the most commonly used. It can be used both for encryption and for digital signatures. RSA computation takes place with integers modulo $n = p \times q$, for two large secret primes $p$ and $q$. To encrypt a message $m$, it is exponentiated with a small public exponent $e$. For decryption, the recipient of the cipher-text $c = m^e \pmod n$ computes the multiplicative reverse $d = e^{-1}(\pmod{(p-1)\times(q-1)})$ (we require that $e$ is selected suitably for it to exist) and obtains $c^d = m^e \times d = m \pmod n$.

The private key consists of $n, p, q, e, d$ (where $p$ and $q$ can be forgotten); the public key contains only of $n, e$. The problem for the attacker is that computing the reverse $d$ of $e$ is assumed to be no easier than factorising $n$ [39].

The key size (the size of the modulus) should be greater than 1024 bits (i.e. it should be of magnitude $10^{308}$) for a reasonable margin of security. Keys of size, say, 2048 bits should give security for decades [42].

Dramatic advances in factoring large integers would make RSA vulnerable, but other attacks against specific variants are also known. Good implementations use redundancy in order to avoid attacks using the multiplicative structure of the cipher-text. RSA is vulnerable to chosen plain-text attacks and hardware and fault attacks. Also, important attacks against very small exponents exist, as well as against partially revealed factorisation of the modulus.

The proper implementation of the RSA algorithm with redundancy is well explained in the PKCS standards (see definitions at RSA Laboratories [43]). The RSA algorithm should not be used in plain form. It is recommended that implementations follow the standard as this has also the additional benefit of inter-operability with most major protocols.

III. STATEMENT OF THE PROBLEM

We will provide two variants of our method. These variants differ on what constitutes a record but are practical and effective privacy-preserving methods for the mining of association rules. We will achieve this by later describing a protocol COMBINE WITHOUT OWNER for the seemingly simple task of bringing the data of all parties together while removing what record was contributed by whom.

The task of mining association rules over market basket data [44] is considered a core knowledge discovery activity since it provides a useful mechanism for discovering correlations among items belonging to customer transactions in a market-basket database. Let $D$ be the database of transactions and $J = \{J_1,\ldots,J_n\}$ be the set of items. A transaction $T$ includes one or more items in $J$ (i.e., $T \subseteq J$). An association rule has the form $X \rightarrow Y$, where $X$ and $Y$ are non-empty sets of items (i.e., $X \subseteq J, Y \subseteq J$) such that $X \cap Y = \emptyset$. A set $X \subset J$ of items is called an itemset. The support $\text{sup}_p(X)$ of an item (or itemset) $X$ is the percentage of transactions from $D$ in which that item or itemset occur in the database. In other words, the support $s$ of an association rule $X \rightarrow Y$ is the percentage of transactions $T$ in a database where $X \cup Y \subseteq T$. The confidence or strength $c$ for an association rule $X \rightarrow Y$ is the ratio of the number of transactions that contain $X \cup Y$ to the number of transactions that contain $X$. An itemset $X \subseteq J$
is frequent if at least a fraction $s$ of the transaction in a database contains $X$. Frequent itemsets are important because they are the building block to obtain association rules with a given confidence and support. The distributed mining of association rules over horizontally partitioned data consists of sites (parties) with homogeneous schema for records that consists of transactions.

Our first solution uses the protocol COMBINE WITHOUT OWNER to bring all transactions together and then let each party apply an association-rule mining algorithm (A-priori or FP-tree, for example) to extract the association rules. This approach is reasonably secure for some settings, but parties may learn about some transactions on other parties. Ideally, it is desirable to obtain association rules with support and confidence over the entire joint database without any party inspecting other parties transactions [9]. Computing association rules without disclosing individual transactions is possible using some global information. For example, if one knows that 1) $\{A, B, C\}$ is a global frequent itemset, 2) the local support of $\{A, B\}$ and $\{A, B, C\}$ and 3) the size of each database $DB_i$, then one can determine if $AB \Rightarrow C$ has the necessary support and confidence since

$$sprt(AB \Rightarrow C) = \frac{\sum_{i=1}^{N} \text{Local Support at site}_i(\{A, B, C\})}{\sum_{i=1}^{N} \|DB_i\|},$$

$$sprt(\{A, B\}) = \frac{\sum_{i=1}^{N} \text{Local Support at site}_i(\{A, B\})}{\sum_{i=1}^{N} \|DB_i\|},$$

and

$$\text{confidence}(AB \Rightarrow C) = \frac{sprt(AB \Rightarrow C)}{sprt(\{A, B\}).}$$

Thus, to compute distributed association rules privately, without releasing any individual transaction, our method adapts Chang et al [45] in a similar fashion as was adapted before for privacy preservation [9, 10]. This is our second more sophisticated and secure approach.

1) Parties compute individually their frequent itemsets at the desired percentage of support. Note that any globally frequent itemset will need to be found by at least one party. The algorithm correctness for finding all globally frequent itemsets is established as an itemset has global support above the global support at $p$ percent only if at least one party has that itemset as frequent in its database with local support at least $p$ percent.

2) For all those itemsets that are above the desired relative support, the parties use the protocol COMBINE WITHOUT OWNER to share records that consist of the local frequent itemsets (and never transactions). The parties also share the size of their local databases. The parties learn the union $U$ of all locally frequent itemsets.

3) $P_i$ recomputes the local count of all the itemsets in $U$.

4) For each itemset in $U$, the parties engage in SMPC of the sum of the local counts, and testing against the minimum required support. Those itemsets that have global frequency are announced to all parties.

In the protocol the parties find global frequent itemsets in rounds of increasing cardinality as it happens with the A-priori algorithm. That is, the parties execute the steps above to find first the local frequent itemsets of size 1 in order to find global frequent itemsets of size 1. This enables them to find global candidate itemsets of size two. Repeatedly, the parties find local frequent itemsets of size $k$ using information on global frequent sets of size $k - 1$ (and thus eliminating some candidates). They share local frequent itemsets of size $k$ with others for obtaining global itemsets of size $k$, and this enables them to formulate global candidate itemset of size $k + 1$.

We would like to emphasise two important aspect of our solution in this application. The first is that, because the protocol COMBINE WITHOUT OWNER does not require commutative encryption, these solutions does not require commutative encryption. The second is that, because the protocol COMBINE WITHOUT OWNER is more efficient, we require $p$ exchanges of encrypted data between the parties less than the previous algorithms for this task.

IV. PROTOCOL TO JOIN DATABASES

We now describe the problem that COMBINE WITHOUT OWNER solves. Let $\mathcal{P} = \{P_0, \ldots, P_p\}$ be a set of $p$ parties where $|\mathcal{P}| \geq 3$. Each party $P_i$ has a database $DB_i$. We assume that parties running the protocol are semi-honest. The goal is to share the union of $DB_i$ as one shuffled database $DB_{Comp} = \bigcup_{i=0}^{p} DB_i$ and hide the link between records in $DB_{Comp}$ and their owners (in the simple version, the records are transactions, but in the second, more secure version, the records are the itemsets; represented for example, by a characteristic vector).

Our protocol COMBINE WITHOUT OWNER employs a public-key crypto-system algorithm on the horizontally partitioned data among three or more parties. In our protocol, the parties can share the union of their data without the need for an outside trusted party. The information that is hidden is what data records where in the position of which party. The union is computed and duplicates are removed. Our protocol is described with one party as the protocol driver, we call this first party Alice.

1) Alice generates a public encryption key $k_{PA}$. Alice makes $k_{PA}$ known to all parties (for illustration we use another two parties called Bob and Carol).

2) Each party (including Alice) encrypts its database $DB_i$ with Alice’s public key. This means, the encryption is applied to each row (record) of the database. Parties will need to know the common length of rows in the database. We denote the result of this encryption as $k_{PA}(DB_i)$ although the encryption is applied per row. Note that, by the properties of public crypto-systems, only Alice can decrypt these databases. But if Alice and Bob hold an identical record $r$, if Alice passes $k_{PA}(r)$ to Bob among the encryption of other records, because Bob also holds $k_{PA}(r)$, Bob would recognise the duplicate record, although not the contents of the row.
3) Alice passes her encrypted rows \( k_{PA}(DB_1) \) to Bob. Bob cannot learn Alice’s rows, since he does not know the decryption key (Fig. 1).

4) Bob mixes his rows with Alice’s rows, removing duplicates. That is, he produces a random shuffle of \( k_{PA}(DB_1) \) and \( k_{PA}(DB_2) \) before passing all this shuffled rows to Carol (see Figure 1).

5) Carol adds and shuffles her rows \( k_{PA}(DB_3) \) to the rows received from Bob but removes duplicates (see Figure 2).

6) The protocol continues in this way, each subsequent party receives a database with the encrypted and shuffled rows of all previous parties in the enumeration of the parties. The \( i \)-th party mixes randomly its encrypted rows \( k_{PA}(DB_i) \) with the rest, removing any duplicates in its rows, and passes the shuffled rows to the \( (i+1) \)-th party by a permutation only know by the \( i \)-th party.

7) The last party (in our illustration Carol) passes the transactions back to Alice (see Figure 2).

8) Alice decrypts the complete set of rows with her secret decrypt key. She can identify her own rows. However, Alice is unable to link rows with their owners because rows were shuffled by each party in the pipeline.

9) Alice publishes the union of the rows to all parties. If the number of parties is \( p \), then \( p-1 \) of the parties need to collude to associate data to their original owners (data suppliers). This is more secure than previous algorithms for this task. It is important to note that our protocol in not completely secure in the sense of multi-party computation. The \( i \)-th party learns how and which of its locally frequent itemsets are also frequent among the previous \( i-1 \) parties. It has been argued that this type of information is innocuous (as previous algorithms for the mining of association-rule mining of horizontal data also reveal this type of information [9], [10]). In fact we leak less information that previous algorithms, since for this situation to happen, the itemset found as a duplicate has no identifiable owner.

To compute the global frequency on an itemset, where each party has the local count we can use previous methods [9], [10] or simply the first party adds a randomly generated number \( r \) to its local value, passes it to the next party. Each party can not learn anything because of the masking effect of \( r \), but adds its local value and passes it to the next. When it arrives back to the first party, it can subtract \( r \) and since the public support threshold is known, release the outcome. The role of the first party can be rotated among the parties.

V. COST OF ENCRYPTION

The theoretical analysis indicates that the computational complexity of RSA decryption of a single \( n \) bit block is approximately \( O(n^3) \), where \( n \) denotes both the block length and key length (exponent and modulus). This is because the complexity of multiplication is \( O(n^2) \), and the complexity of exponentiation is \( O(n) \) when square and multiply is used. However, an illustration to the efficiency of RSA is in the OpenSSL implementation for secure, authenticated communication between different sites [46]. SSL is short for Secure Sockets Layer, a protocol developed by Netscape for transmitting private data via the Internet. The overhead of SSL communication has been found of practical affordability by other researchers [47].

We analysed the cost of RSA encryption in terms of computation, number of messages, and total size. For this analysis, we implemented RSA in Java to calculate the encryption time of a message of size \( m = 64 \) bytes with encryption key of 1024-bits. This time was 0.001462 sec. on a 2.4MHz Pentium 4 under Windows. This is perfectly comparable with the practical computational cost suggested by earlier methods [9]. While some regards RSA as too slow for encrypting large volumes of data [48], our implementation is particularly competitive. An evaluation of previous methods [9] suggested that (on distributed association rule mining parameters found in the literature [49]), the total overhead was approximately 800 seconds for databases with 1000 attributes and half a million transactions (on a 700MHz Pentium 3). Our implementation requires 30% of this time (i.e. 234.2 seconds), but on a Pentium 4, in any case, perfectly affordable.

VI. CONCLUSION

We have proposed a flexible and easy-to-implement protocol for privacy preserving data sharing based on a a public-key crypto-system. The protocol is efficient in practical settings and it requires less machinery than previous approaches (where commutative encryption was required). 1, the sharing process was reduced from \( 3p \) steps to \( 2p \) steps. Commutative encryption can be implemented with RSA; however, all schemes

---

1 Commutative encryption means that if we have two encryption algorithms \( E_1 \) and \( E_2 \), the order of their application is irrelevant; that is \( E_1(E_2(x)) = E_2(E_1(x)) \).
are not only more expensive that what we proposed here, but commutative encryption has a probability/certainty factor that requires delicate management. Our methods here are safe from this. Thus, our methods ensure that no data (transactions) can be linked to a specific user. The protocol allows users to conduct private mining analyses without loss of accuracy (as opposed to data sanitization). Our protocol works under the common and realistic assumption that parties are semi-honest, or honest but curious, meaning they execute the protocol exactly as specified, but they may attempt to infer hidden links and useful information about other parties. The requirement for non-collusion is higher than previous methods, since $p-1$ parties must collude to compromise the protocol. We leak some information, but this is less that what previous methods have regarded as innocuous and previous research has explored whether parties are willing to trade off the benefits and costs of sharing sensitive data [50], [51]. The results of this research showed that parties are willing to trade-off privacy concerns for economic benefits. There are few issues that may influence practical usage of the presented protocol. The issue of having a party that can be trusted with shuffling the records and publishing the database to all other parties can be solved with slightly more sophistication, but essentially no overhead; if there are $p$ parties, each party plays the data distributor with $1/p$ share of the data, and we conduct $p$ parallel rounds.

We also showed that our protocol is efficient and especially more efficient than previous protocols [9]. We showed that by using our protocol not to share entire local databases, but local itemsets, we can mine privately association rules on the entire data without revealing any single transaction to other parties.

ACKNOWLEDGMENT

The authors would like to thank the Australian Research Council for its support.

REFERENCES


